CACTUS: Automated Tutorial Course Generation for Software Applications

Federico García
Escuela Técnica Superior de Ingeniería Informática
Universidad Autónoma de Madrid (UAM)
Cantoblanco, 28049, Madrid (Spain)
Phone: +34 91 348 22 91
Federico.Garcia@ii.uam.es

ABSTRACT

Novice users often face many difficulties in mastering current highly interactive systems. In this paper we describe CACTUS, an interactive system used to develop tutorial courses for software applications. CACTUS tutorial courses provide more adequate and more dynamical explanations than currently existing teaching components, since they are task-oriented and provide just-in-time context-dependant explanations. These tutors are also able to follow-up the user activity and act according to what they perform. CACTUS is an environment that uses the model-based design technology. In particular, CACTUS uses declarative hierarchical task-models to derive guidance instructions. Additionally, CACTUS releases tutorial course designers from part of the intensive workload of developing tutor programs as these guidance components are currently developed. This system helps to generate application tutorial courses based on a metaphor that represents the contents of the courses as if they were textbooks, so that learning an application is assimilated to reading a book on certain subject and performing some activities.

Keywords

Tutorial Course Generation, User-Task Models, Programming by Demonstration.

1. INTRODUCTION

Current graphic interfaces enclose a complexity that was unthinkable not many years ago. These systems have in many cases hundreds of commands, often operating in a different way depending on the context. Thus, only few advanced users can take advantage of a high percentage of the power of these systems. However, these tools rarely come together with systems that let the users learn systematically how to use their applications. This fact is especially dramatic in the case of novice users.

In this paper we introduce CACTUS, which stands for *Creating Application Courses about Tasks Using Scenarios*, an environment aiming to palliate the problems related to current systems for software tutoring. CACTUS offers an integrated environment to design tutorial courses for interactive applications, and includes the necessary support to allow the system to follow up the pupils during the tutorial course execution.

From the point of view of the tutorial courses generation, CACTUS offers a framework for the development of teaching activities. This framework allows designers to create and modify easily the courses in a fully interactive way, by using both visual and demonstrational techniques [4]. This permits the specification for the tutorial course by using the same application for which we are creating it.

The produced tutorial courses are supported by the development technology based on declarative interface models [18], that provide many benefits in the application development [15], as model reuse, rapid application prototyping, and so on. Furthermore, the information in those models makes it possible for application-external tools to reason about the application state at run-time, in order to modify the interface behavior. In this case, these models make it possible for CACTUS to give the final users, the pupils, the fundamental feature of receiving feedback from the following-up of their activity and the evaluation of their knowledge on the explained tasks.

The CACTUS environment has been tested to generate tutorial courses for some applications. First, it has been proved with interactive interfaces to teach continuous digital simulation systems such as the solar system and Volterra equations [1]. It has also been tested to generate a tutorial course to teach OOPI-TasKAD, an object-oriented computer aided design environment based on the prototype/instance paradigm. Furthermore, tests have taken place on an electronic agenda and, finally, we have tested our tool to generate a course for *Schoodule*, an application that uses a database and a constraint solver to generate school schedules.

This paper is structured as follows. First, we provide the motivation for the system by putting it into context. Then, the system architecture will be introduced, describing the subsystems CACTUS is based on. Afterwards, we will give a more detailed description of CACTUS tutorial courses and their operation, both during the course design and the execution. Finally, we will provide some conclusions and future research lines.

2. CONTEXT AND RELATED WORK

Even today, the most common guidance systems are based on hypermedia explanations, describing how to access each command and the functions they are supposed to perform. This format has serious drawbacks [3]. First, the explanations are given at a very low level, only referring graphic interface objects, and they never refer to higher conceptual level tasks. Second, there is an isolation of the guidance component with respect to the application, and there is no interconnection between both components. Third, the processes are fully explained before the user begins to carry out the task, so s/he has to switch between the application and the help windows to read the explanations and perform the task in parallel. Finally, there is no feedback from the help system towards the user to indicate when any accomplished step is not a correct one. As a consequence, the guidance component is passive and unable to correct the user.

A few applications deliver tutor systems to guide the users through the application learning. These tutors have some drawbacks that make them difficult to be widely used. First, those tutors usually simulate the application behavior so that the users have the feeling that they are using the real system, not a simulated one. Even though this *application clone* replicates only some system behaviors, its development implies very high costs. Second, those tutors do not usually teach by using each user's work context, but they teach by means of predefined examples. Although the provision of these examples may help in some cases, many times users want to use their own work contexts. Finally, there are high maintenance costs, since each change in the application should be reflected in the tutor component to guarantee the soundness of the explanations.

From the research point of view, in the last years a great effort has been devoted to generate help, automatically or semi-automatically, for interactive applications using different paradigms. Some systems such H3 [10] are data oriented. Some others, such the works based on Petri nets [12] are application-state oriented, and others are action-oriented [17]. We believe that these kinds of guidance are not adapted enough from the user's point of view. Moreover, the type of guidance provided by these systems is not adequately structured to be assimilated by novice users in general. Nevertheless, hierarchical user-task models give us the possibility of both incorporating the user's point of view about the application and structuring the explanations in an understandable way. Next, we will remark the most relevant task-oriented guidance systems.

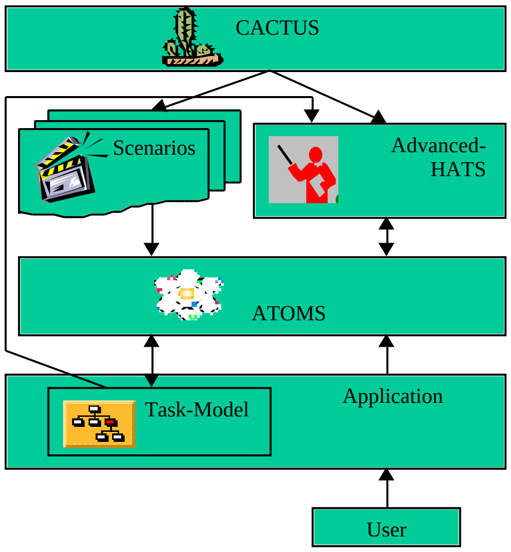
Pangoli and Paternó [13] were pioneers in offering help from user task models. Thus, the information the user receives has great semantic power. However, the work they propose does not follow-up user activity, and it does not offer feedback about the task accomplishment.

Teach me While I Work [2], TWIW, also produces task-oriented guidance, including not only help but teaching capabilities. In this sense, it is able to filter incorrect user interactions and provide users with feedback about their actions. One of the biggest limitations of this system is that the task model it is based on does not include any contextual information. Moreover, this system is mostly oriented to users who have some familiarity with the applications they want to learn, but it is not intended to be used by novice users.

Finally, Help for ATOMS Task System, HATS [5], is a help system that uses user-task models to generate the explanations. It has been later endowed with similar teaching capabilities to those of TWIW, as we will see when we describe the Advanced-HATS tutoring system. In particular, HATS provides the user with dynamic help, updating the help messages according to what s/he needs to perform at each moment. The most novel feature of this system is its ability to offer graphical feedback, by highlighting application graphical objects. This feedback indicates relevant information about what has been previously done and what is still to be done. HATS is based on the task models used by ATOMS [6][16], a system that manages user-task models at run-time and permits the tasks to incorporate and manage contextual information, in the form of parameters.

3. ARCHITECTURE

The general architecture of the environment introduced in this paper is shown in **Error! Unknown switch argument.**. This figure shows how CACTUS takes advantage of a refinement of the HATS help system, a tutoring system called Advanced-HATS, and a *scenario execution module*, to provide course designers with an environment with which, at the lowest costs, they provide novice users with tutorial courses for their applications. Both modules, Advanced-HATS and the *scenario execution module*, are built on top of the ATOMS run-time task management system in order to get task-oriented support. As it will be show in Section **Error! Unknown switch argument.**, the nature of the indications provided for the users is task-oriented, so they get information that is more comprehensible for them that it would be if they were based on other nature models. CACTUS allows the designers to specify the previous *scenarios* that may be used as context for the teaching of user-



tasks.

Figure Error! Unknown switch argument.**: System architecture**

3.1 Managing Tasks: ATOMS

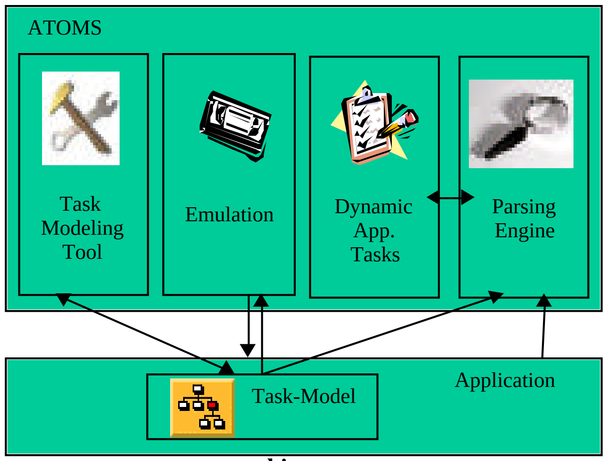
ATOMS, Advanced Task-Oriented Management System, is a run-time user task-model management system. It is not a system aimed to design application user-interfaces from scratch, either by automating the generation or by specifying the interfaces, but it is aimed to allow other processes to request to be notified when tasks are completed and invoke application tasks. These facilities support the constructions of agents that can assist users in various ways.

ATOMS-based applications define hierarchical representations of the user-tasks they provide support to. These representations, called user-task models, are composed of the application user-tasks and rules linking those tasks.

There are two kinds of tasks: atomic ones and composed ones. The atomic tasks model the actions a user can accomplish directly by interacting with the application user interface, that is, these tasks are linked to the application by means of descriptions of the corresponding *abstract interaction objects*. The role of the composed ones is to allow the modeling of high-level abstract tasks. Both types of tasks may have associated parameters, acting as contextual information for them. On the other hand, each rule relates a particular composed task with certain set of atomic and composed tasks. Furthermore, each rule includes some information such as the execution relationship between subtasks –sequence, parallel, xor-, the parameter flow between tasks –how low-level task parameter values are converted into high-level parameter values-, which tasks may be optional and/or multiple and under which conditions, and which pre- and post-conditions must hold for a subtask execution. In this sense, tasks and rules in this system act as Unification Grammars in Natural Language Processing systems [9].

ATOMS task models are specified using a declarative modeling language which incorporates inheritance both for tasks and rule definitions, in order to improve the reuse of partial models. Applying this inheritance concept, a task can be seen as a prototype for other task instances, which may be more general or more specific than their prototype. Rule inheritance allows designers, starting from a certain relationship between several tasks, to relax the conditions of such relationship or to turn them more rigid according to their use context at each moment. This specialization may have two goals: from a rule prototype, we can define another one substituting thoroughly the first one; or we can have a more specific rule that has preference over its prototype to be applied, but that does not eliminate it for the cases in which it could not be applied.

Figure Error! Unknown switch argument.**: ATOMS**



architecture

ATOMS functional core is composed by four main blocks (see **Error! Unknown switch argument.**): the *Parsing Engine*, the *Dynamic Application Tasks*, the *Task Modeling Tool* and the *Emulation* module. The first module follows-up the user activity while interacting with the application, in order to recognize at any moment which tasks s/he is carrying out. The *Dynamic Application Tasks* represents both the state of the active tasks and a task historic, in order to allow external tools to reason about users performance and to anticipate their actions. The third module is devoted to interactive generation of ATOMS task models. For a detailed description of these modules, see [7].

For our purposes, the most important module is the *Emulation* one. This module executes tasks using animations, and it is provided as a service for external value-added tools to modify the application interface behavior at run-time. This module is capable of emulating the user interactions by analyzing the tasks and rules descriptions specified in the application task model. Starting from the model, the requested task and the provided parameter values, it decides which atomic tasks have to be executed, the parameter values and their sequencing. For example, this module could execute the instruction *SelectTeacher name "John Dale"* using animations, as if the user accomplished it. *Emulation* also executes requests where some associated parameters do not have any value assigned. *Emulation* treats this situation by indicating the user to provide interactively the parameter values and, also, it offers graphical feedback to the user on different available options, if possible.

3.2 Managing Tutorial Courses: CACTUS

CACTUS is a system for the interactive development of tutorial courses. CACTUS provides more adequate and more dynamical explanations than currently existing tutors and follows user activity to act accordingly. Additionally, CACTUS releases the tutorial course designers from part of the intensive load of developing the tutors. The CACTUS architecture is based on two modules (see **Error! Unknown switch argument.**). The first one, Advanced-HATS, is a tutoring system to teach the users how to accomplish a certain task, while the second one, the *scenario execution module*, prepares appropriate contexts for the teaching activities.

Advanced-HATS is a refinement of the HATS help system, whose main improvements are centered on the incorporation of tutoring capabilities to the help ones mentioned in the previous section. In addition to the HATS features, Advanced-HATS may act with several degrees of flexibility. Thus, the system decides when interactions are to be filtered and when to force the user to follow a correct path, depending on the degree of flexibility. Another feature of this subsystem is the possibility of teaching parameterized tasks. Advanced-HATS provides messages referring the task parameter values and filters actions that are incorrect accordingly to the values adopted by the task parameters, thus forcing the user to perform the task having the indicated parameter values. A detailed description on how HATS gives dynamic guidance and how it uses the information in the task models to generate automatically the messages can be found in [5].

The *scenario execution module* prepares appropriate contexts to carry out the teaching of the tutorial course tasks. A *scenario* is a procedural description of a process, including references to application tasks, variables, conditional blocks, loops, and so on, and it is interpreted by the *scenario execution module*. For example, let us suppose that in a CAD application we would like to teach a user how to lift a design from 2D to 3D. Then, it would be reasonable to prepare a *scenario* that would show the pupil the creation of a kitchen floor, so that afterwards s/he would learn how to transform that 2D design into its corresponding 3D design. The task references in these descriptions play the same role played by system calls in other environments. That is, the interpreter, basing on the ATOMS' *Emulation* module, executes the referenced task by using animations.

Nevertheless, to provide designers with the highest flexibility, CACTUS allows users to practice the tasks in their own work context instead of using predefined *scenarios*. Thus, CACTUS can be set with three different behaviors: always execute the *scenarios*, never do it, or always ask pupils before executing a *scenario* if s/he wants it to be executed.

3.2.1 Features

CACTUS integrates designer oriented services, such as support for creating, editing, testing and debugging tutorial courses, and pupil-oriented services, such as support for the execution of the courses. CACTUS has two operation modes: the *execution mode* permits to execute, test and debug the courses, and the *edit mode* allows to create interactively the tutorials and to modify them.

In our system, a course is composed by units, each one teaching a set of user tasks related by some criterion. Although CACTUS courses can be specified through a programming language, as we will explain later, the CACTUS *edit mode* allows designers to create interactively the tutorial courses and to modify them. To achieve this goal, CACTUS uses the metaphor of representing a tutorial course as if it were an interactive textbook, and associates the most important parts of the tutorial course with representative parts of a book. Using direct interaction procedures can modify these interactive books, since CACTUS incorporates visual and demonstrational techniques [4]. The general operation of CACTUS is shown in **Error! Unknown switch argument.**. We can see that neither users nor designers need to access directly the contents of the tutorial courses, but they simply interact with the CACTUS interface to do it.

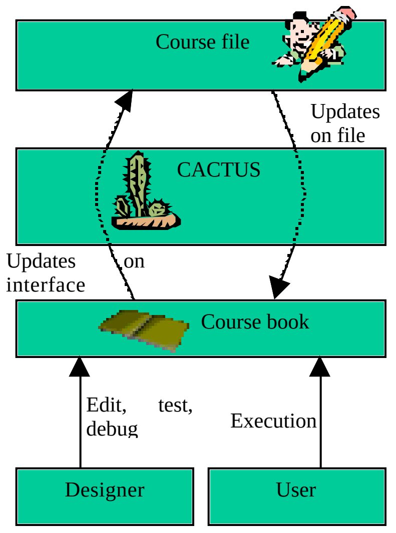
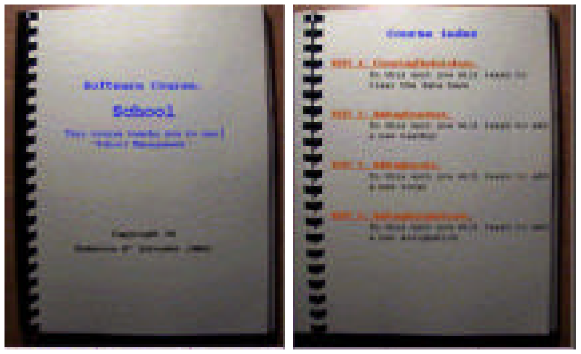


Figure Error! Unknown switch argument.**: CACTUS general operation.**

Along this section we will illustrate some CACTUS features using examples from a course for *Schoodule*, a system that uses a database and a constraint solver to help on the generation of school schedules. *Schoodule's* database essentially reflects four entities and a relationship. The first entity corresponds to schoolteachers. A second entity describes the different subjects in the school. The information dealing with the groups is reflected in another entity, and the last one deals with the available classrooms. Finally, the *assignment* relationship reflects which teacher imparts each subject for each group of pupils. From all this information, *Schoodule* generates a schedule proposal, indicating both the timetable and the classrooms for each lecture.

3.2.2 Book Metaphor

CACTUS visualizes and allows the modification of the courses through a representation of the tutorial courses as if they were interactive textbooks. Most parts of the books are automatically generated by CACTUS, and the designers only have to include the desired contents for the pedagogical units. The main feature in our courses with respect to those generated with existing systems is the fact that our courses follow user activity. This makes possible to provide users with context-dependant explanations. Thus, the user does not learn by reading the book contents, but s/he learns by executing them. This is complemented by offering the use of predefined *scenarios* as the context for the learning process, in addition to using their own work context. Another important feature is the automatic incorporation of hypertext links, so that it is easy to navigate through the books and the designer does not need to worry about navigation issues. Users can also navigate along



a book by using the classic *Next* and *Previous* commands. In the rest of this section we will give a description of the structure of a CACTUS book.

Figure Error! Unknown switch argument.**: A tutorial course book cover and index.**

First, CACTUS generates a cover like the one shown in **Error! Unknown switch argument.** (left). This cover includes the course title and a general description to provide an overall idea of the course purpose. Course designers can customize these fields by in-place editing them.

Afterwards, CACTUS automatically generates an index of the course pedagogical units using the same order the designer follows to develop it, as shown in **Error! Unknown switch argument.** (right). Each entry in the index contains a customizable unit title and a short description of its contents. For each entry, the system generates a hyperlink to the starting page of the corresponding pedagogical unit. The contents of a pedagogical unit will be explained later.

One of the main characteristics of a CACTUS tutorial course is that it can be followed in several ways. In this sense, our interactive books are similar to those textbooks used in advanced courses, which usually have a predefined order to be read, but they can be read following other orders, since the matters covered do not have linear relationships. CACTUS tutorial courses include this sequencing notion between the different pedagogical units of the tutorial course. CACTUS automatically manages the sequencing, and allows the user to execute a unit depending on whether the previous units have already been taught or not. The ability to follow user interaction allows the system to manage automatically the unit sequencing. The next two parts of the book are narrowly related to this sequencing notion.

First, a directed graph representation of the tutorial course structure is generated, as shown in **Error! Unknown switch argument.** (left). Each node in the graph represents a pedagogical unit, and its color represents its execution state. The state of a unit indicates if it has been already executed, if the unit has not been executed yet but it is possible for the user to execute it, or if the unit is currently blocked because some of its predecessor units have not been accomplished yet. An arrow from node A to node B indicates that the unit associated to A has to be performed before performing the unit associated to B. The pedagogical unit states are automatically updated by CACTUS as the user follows the tutorial course.

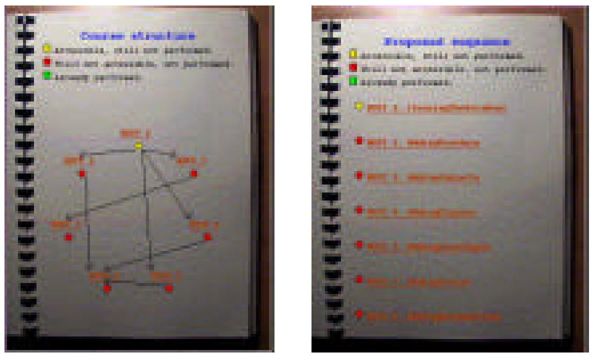


Figure Error! Unknown switch argument.**: Tutorial course book graph and proposal.**

Second, a proposal of a sequencing of units for the course is generated, as shown if **Error! Unknown switch argument.** (right). In this proposal, when a node is visited all its predecessors have already been visited, so that it will never be the case that the user tries to access a blocked unit. This proposal and the directed graph representation are automatically generated and managed by CACTUS, so the designer does not have to worry about them.

Afterwards, as many chapters as pedagogic units in the tutorial course are presented, as shown in **Error! Unknown switch argument.** (left). Each unit incorporates some task-tutoring sessions and, probably, some examples and *scenarios* to execute. The page layout includes explanations about the task-tutoring sessions, the examples presented, the *scenarios* to be executed, and images. These explanations, that indicate the users the different things to do during the unit, are automatically generated by CACTUS from the tasks to be taught, the examples to be executed and the *scenarios* to be prepared, as it will be explained later.

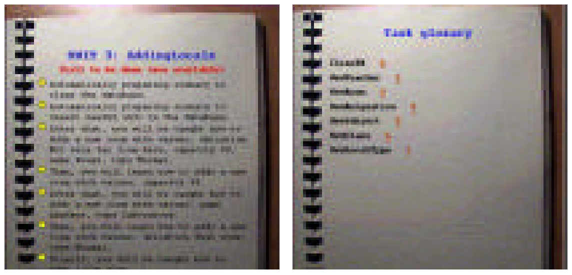


Figure Error! Unknown switch argument.**: Tutorial course book units and task glossary.**

Finally, CACTUS automatically generates a task glossary at the end of the book, as shown in **Error! Unknown switch argument.** (right). In this glossary, an entry for each task being taught in the course can be found. Each entry relates a task with the list of units containing tutoring about it. Each item includes a hyperlink to the unit page it refers to. Then, the process of searching a certain task is reduced to a look-up process in this glossary and to follow a hyperlink. This glossary is managed by CACTUS transparently, so the designer does not have to worry about updating it during the development phase.

In the next two subsections CACTUS will be described accordingly to its two operation modes: the *execution mode* and the *edit mode*.

3.2.3 Tutorial Course Execution

In the *execution mode*, CACTUS allows executing some parts of the interactive books to teach the user how to perform some user tasks by means of an interactive software application. Course designers can also use this mode to test and debug the courses at development-time.

Some sections of the book have a group of instructions associated. Instructions in this environment can be classified in three groups. The first one includes those for specifying task-teaching activities, telling the system to show an example of a task, preparing suitable *scenarios* to practice with, and showing users information about their performance. This kind of instructions has an immediate feedback on the application interface. The second ones do not have an immediate feedback towards the user, and includes low-level specifications such as variable assignments or instructions aiming to format the course (unit descriptions, unit titles, images, and so on). This group also includes control structures such as loops, conditional blocks or random execution blocks. Finally, a third group controls how the *scenario execution module* and Advanced-HATS operate, including setting the degree of flexibility for the tutoring module, the quantity of messages to be shown during the teaching activities, the *scenario* execution speed, the *scenario* execution refreshment rate, and similar ones.

The executable sections of an interactive tutorial course book are the pedagogical units and the proposal section.

The pages of the pedagogical units usually contain task-teaching activities, examples of predefined tasks, *scenario* preparation instructions and feedback messages for the users. These kind of instructions are adequately related by the use of different types of control structures.

For example, in our course for *Schoodule* there is a unit that explains how to add a new teaching *assignment*. Let us see which the contents of this unit are and what its dynamic behavior is. In this unit, the first thing the tutorial course includes is the preparation of a *scenario*. This *scenario* adds some default teachers, subjects, groups and classrooms. In this way, we are sure the user will not have any problem during the practice, because of a lack of previous information in the *Schoodule* database. Executing this first step of this unit will cause CACTUS to ask the *scenario execution module* to prepare the *scenario*, in order to add default information to the application database. Of course, the user may choose to use her/his own work context instead of preparing the *scenario* provided by the designer. After this, the unit includes a teaching activity. This will teach the user how to insert a new teaching *assignment* to the *Schoodule* database. This teaching activity will be done under the supervision of the Advanced-HATS tutoring component, which will explain the task to the user as explained in [5] and will tell her/him whether her/his actions are correctly performed. Once the user would have successfully inserted a new teaching *assignment*, the tutorial course will give the user some feedback about the task that has been just performed, by means of a feedback message that includes the parameter values of the new teaching *assignment* added. In addition, the unit provides the user with the possibility of receiving examples about the *add assignment* task, using different parameter values, at any moment. In this area of examples, the designer includes as many examples as s/he desires. These examples will be executed by using the ATOMS' *Emulation* module, that is, by using animations to perform the tasks. Moreover, the user decides when s/he wants the system to execute the proposed examples. The effects of these examples may be undone if the application provides support for the multiple undo feature.

When the user adequately accomplishes the pedagogical contents of a unit, CACTUS automatically updates the state of that unit, then enabling the access to its succeeding chapters, if appropriate. When a unit is disabled, CACTUS will not permit pupils to execute it, they can only inspect the contents. CACTUS shows under each unit title its state: *'Blocked unit*, *'Already Performed'*, '*Accessible Unit'* or '*Being Performed'*. During the execution of a unit, CACTUS highlights at each moment the instruction being accomplished, so that the user always knows how much s/he has already performed and how much remains to be accomplished.

As previously mentioned, the proposal page is also executable, and its execution means the successive execution, according to the order being shown, of the pedagogical units in the course. By executing the proposal page, CACTUS makes the unit precedence graph transparent to the user, so that s/he will never try to execute a unit whose execution is blocked.

CACTUS also has the option of saving the execution state of a course to reload it later on.

3.2.4 Interactive Creation and Modification

The CACTUS *edit mode* aims to help designers on creating tutorial courses from scratch and to modify existing ones.

From the pupils' point of view, our courses are just like interactive books including advanced features that teach them how to perform some user tasks and follow their accomplishment of those activities. This is the same case for novel course designers, who only deal with CACTUS interface when they want to create a course.

A CACTUS course is defined in a text file through a programming language, executed by an interpreter when the system gets into the *execution mode*. Thus, only advanced tutorial course designers need to access this textual tutorial course representation.

CACTUS is in charge of the layout of the interactive books, including not only the automatic generation of most parts of the books, but also the general layout of their pages. The system generates the interactive books and, by default, adjusts the different parts of the pages in order to wrap words around figures, scale images, update dynamic graphics or generate hypertext links.

This section aims to describe how CACTUS courses are built. The operations to specify the courses can be reduced to addition and deletion of instructions. To eliminate instructions, the designer selects the parts to be eliminated and presses the *Supr* key. To add new ones, it depends on the type of instructions, as it will be seen in this section.

We can insert tutoring activities in two ways. The first one uses the *'Insert tutoring session'* command. Then, the designer selects the desired task from a list of all the tasks in the model. After that, the designer will be asked if s/he wishes that some parameter values make a particular condition held. The formulas to compute these conditions may include constant values, runtime computed expressions or random values generated by CACTUS. For example, the designer can select the *AddTeacher* task and force the user to add a teacher named '*John'*. The other way of adding a tutoring session is to use the *'Insert tutoring about...*' command, once the application is started. This command makes CACTUS watch and record every task the designer performs on the application. During the process, CACTUS inserts as many task tutoring instructions as tasks are performed by the designer until s/he uses the *'Finish inserting tutoring'* command. For example, if the designer adds a new teacher, CACTUS will automatically generate a tutoring instruction for the *AddTeacher* task.

It is worth mentioning that there is independence between the procedures used by the designer to specify the tasks and the procedures that the pupils choose to perform those tasks during the learning process. For example, the designer may use a menu command as a part of a higher-level user task, but the user will be able to push a button to perform the same task. This is due to the fact that CACTUS deals with high-level tasks, not just sequences of atomic ones.

The procedures for adding examples to pedagogical units are similar to the ones used to add tutoring activities.

To introduce a *scenario* execution instruction, the designer may use the *'Insert scenario already made...'* command, which will allow him/her to select the *scenario* to be prepared from any *scenario* library. Alternatively, s/he may use the *'Insert scenario to be defined...'* and *'Finish inserting scenario'* commands. These commands will make CACTUS start watching and recording tasks in a new *scenario*, and finish registering them, respectively. Then, CACTUS inserts a reference to the *scenario* in the course. For example, the designer may add some teachers, classrooms, groups and subjects and CACTUS will include a *scenario* which, when executed, will fill the database with some values.

For other language structures like conditionals or loops, the designer will select the construction to be inserted and, depending on the construction, will provide some parameters on CACTUS requirement. CACTUS analyses the instructions at tutorial course creation time. In this way, most mistakes are identified when they are introduced, thus saving quite an effort to the designer. The rest of the instructions includes pop-up messages with graphical references, static and dynamic images, variable assignments, instructions to control the Advanced-

HATS and the *scenario execution module* operation, and control-flow structures. The static images correspond to bitmaps, while the dynamic ones are instances of application graphical objects and are dynamically updated by CACTUS during the tutorial course execution. Among the instructions to control Advanced-HATS and the *scenario execution module*, it is worth mentioning those that affect the Advanced-HATS flexibility degree, the amount of messages displayed by the tutor, the *scenario* execution speed and the *scenario* execution refreshment rate. Finally, the control-flow structures include loops, conditionals and random blocks.

The designer can also add or eliminate pedagogical units, and interactively edit the sequencing relationships between them. These operations make CACTUS modify some parts of the course book: index, precedence graph, sequence proposal, and glossary, to maintain the consistency along the book parts.

Finally, let us explain how CACTUS automatically generates the explanations in the unit book pages from the contents of the pedagogical units in the course specification. CACTUS generates context-dependant descriptions only for tutoring activities, example executions and *scenario* preparation instructions, and allows editing them interactively. However, the system does not generate descriptions about, for example, setup instructions or control structures, because they are too low-leveled instructions from the user's point of view. For task teaching activities, descriptions are generated like *'Next, you will have to* ' canned with the task description, obtained from the task model. CACTUS describes the examples in the units by canning *'Show me how to* ' with the description of the task involved in the example. For references to *scenario* preparation instructions, the description is obtained by canning the message *'Preparing scenario to '* with the *scenario* description, extracted from the *scenario* specification. Since the automatically generated descriptions use information obtained from some sources, they are rarely appropriate to be directly delivered to the pupil. These generated explanations are usually close to useful descriptions, but often include grammatical inadequacies. Thus, CACTUS allows designers to interactively edit them directly on the book interface, and afterwards save the modified ones. Subsequently, CACTUS offers maximum flexibility and lowest cost, since the automatically generated descriptions release the designers from a great workload.

CACTUS offers a debug window with several capabilities, ranging from tracing the course to allowing a step-by-step tutorial course execution, in order to make the design easier. The designer can also inspect and modify the variable values.

CACTUS also reduces the course maintenance costs to almost the task model maintenance ones, whose specification is accomplished through visual and demonstrational techniques in ATOMS [7]. In addition, CACTUS manages which tasks in the application task model are taught in the course being created and which are not, and it notifies when a unit should be modified because it teaches a task unspecified in the current task model version, and which tasks remain to be taught in the course.

4. CONCLUSIONS AND FUTURE WORK

We have described how the user interface design technology based on declarative models can be used to generate tutorial courses for interactive applications. CACTUS tutorial courses overcome many of the main problems related to the creation of the guidance component of interactive applications with current technologies. CACTUS represents the tutorial courses by assimilating them with interactive textbooks, and uses hyperlinks to make them intuitive and easy to use for the pupils. Furthermore, the tutorial courses add to the textbook capabilities the power of the Advanced-HATS tutoring system in which CACTUS is based on. Thus, the courses are completely interactive and able to follow-up pupils activity. The costs of generating tutorial courses are reduced to the minimum because their development is based on model-based technology and because designers can use an interactive environment with support for programming by demonstration techniques for building them.

CACTUS offers an integrated framework for the development and execution of tutorial courses. This framework includes tutorial course creation oriented services, such as programming by example techniques or support for the course content management; and it also includes execution-oriented services, such as tracing and debugging facilities. CACTUS and all the subsystems it is based on are thoroughly coded in C++. The system implementation uses the AMULET framework [11], which includes an object-oriented system based on the prototype/instance inheritance paradigm. Moreover, a distributed version of the ATOMS subsystem has been implemented for managing work-flows models [8], and a Java version of the system is partially available. This work aims to facilitate the development of tutorial courses for distributed interactive tasks through Internet.

As future work, we are also interested in studying how CACTUS can take advantage of the use of user profile declarative models. This kind of models have been used within some Model-Based Interface Development Environments (MB-IDEs) such as MASTERMIND [19] or MOBY-D [14] to restrict the access to some user tasks or to customize the user interface depending on the profile of the current user. We aim to exploit this kind of model to allow designers to develop tutorial courses customized for each user profile.

This customization will extend the work to several levels. The first one will be the addition of user profiles at the ATOMS level to associate user-dependant restrictions to the task management level. After that, the Advanced-HATS tutoring system should be adapted to exploit user profiles, adapting the messages and choosing different procedures to perform a task depending on the current user profile. The *scenario execution module* should automatically adapt the *scenarios* to the current user profile. Thus, depending on his/her skill level, the system should automatically decide how complex the *scenario* should be, or it could even opt for not executing any *scenario*, allowing him/her to practice directly with his/her own work context.

5. ACKNOWLEDGMENTS

Special thanks to Roberto Moriyón for his extensive remarks on the ideas of this paper. I would also like to thank the anonymous reviewers who contributed in a great deal for the quality of this presentation. This work has been partially supported by the Plan Nacional de Investigación, project numbers TIC96-0723-C02-01/02 and TEL97-0306.

6. REFERENCES

[1] Alfonseca, M., García, F., de Lara, J., and Moriyón, R. "Generación Automática de Entornos de Simulación con Interfaces Inteligentes", Revista de Enseñanza y Tecnología, ADIE, nº 10. December 1998.

[2] Contreras, J. and Saiz, F. “A Framework for the Automatic Generation of Software Tutoring”. In Proceedings CADUI'96, Computer-Aided Design of User Interfaces, Eurographics, Belgium, June 1996.

[3] Contreras, J. “A Framework for the Automatic Generation of Software Tutoring”. Phd. Thesis, Université René-Descartes, Paris, 1998.

[4] Cypher, A. “Watch What I do, Programming by Demonstration”. MIT press (ed. A. Cypher), Cambridge, Ma., USA, 1993.

[5] García, F., Contreras, J., Rodríguez, P. and Moriyón, R. “Help Generation for Task Based Applications with HATS”. In Proceedings EHCI'98, Creta (Greece), September 1998.

[6] García, F., Rodríguez, P., Contreras, J., and Moriyón, R. “Gestión de Tareas de Usuario en ATOMS”. IV Jornadas de Tecnología de Objetos, JJOO'98, Bilbao (Spain), October 1998.

[7] García, F., “Towards the Generation of Tutorial Courses for Applications”. 5th ERCIM Conference on User-Interfaces for All. Dagstuhl (Germany), November 1999.

[8] García, F. and Moriyón, R.: “A Framework for Distributed Task Management”. In Third Argentine Symposium on Object Orientation, ASOO'99, September 1999.

[9] Maxwell, J.T. and Kaplan, R. M. "The Interface between Phrasal and Functional Constraints", Computational Linguistics, no. 4, 1994.

[10] Moriyón, R., Szekely, P. and Neches, R.: “Automatic Generation of Help from Interface Design Models”. In Proceedings of CHI’94, ACM Press, 1994.

[11] Myers, B.A., McDaniel, R.G., Miller, R.C, Ferrency, A.S., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A. and Doane, P. “The AMULET Environment: New Models for Effective User Interface Software Development”. IEEE Transactions on Software Engineering, Vol. 23, no. 6. June, 1997. pp. 347-365.

[12] Palanque, A., Bastide, R. and Dourte, L.: “Contextual Help for Free with Formal Dialog Design”. In 5th International Conference on Human-Computer Interaction, Orlando, Florida, USA. 8-13 August 1993.

[13] Pangoli, S. and Paternó, F. “Automatic Generation of Task-oriented Help”. In Proceedings UIST'95, Pittsburgh, ACM Press, 1995.

[14] Puerta, A.R. , “A Model-Based Interface Development Environment”. IEEE Software, 14(4), July/August 1997, pp. 41-47.

[15] Puerta, A.R. “Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models”. First Annual Workshop On Real-Time Intelligent User Interfaces For Decision Support And Information Visualization, San Francisco, January 1998.

[16] Rodríguez, P., García, F., Contreras, J. and Moriyón, R. “Parsing Techniques for User-Task Recognition”. 5th International Workshop on Advances in Functional Modeling of Complex Technical Systems, Paris (France), July 1997.

[17] Sukaviriya, P. and Foley, J.D: “Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help”. UIST’90, pp. 152-166, 1990.

[18] Szekely, P., Luo, P. and Neches, R. “Beyond Interface Builders: Model-Based Interface Tools”. Proceedings of INTERCHI'93, 1993, pp. 383-390.

[19] Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. and Salcher, E. “Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach”. In Engineering for Human-Computer Interaction, L. Bass and C. Unger (eds), pp. 120-150. Chapman & Hall, 1996.